

- [54] METHOD FOR DISK I/O TRANSFER  
[75] Inventors: Paul L. George, Westboro; David M. Waxman, Framingham; Randall T. Sybel, Blackstone; Elliot H. Mednick, Millbury; Kevin J. O'Brien, Norfolk; Joseph M. Spataro, Ashland, all of Mass.  
[73] Assignee: Prime Computer, Inc., Natick, Mass.  
[21] Appl. No.: 166,515  
[22] Filed: Mar. 9, 1988  
[51] Int. Cl.<sup>4</sup> ..... G06F 13/38  
[52] U.S. Cl. .... 364/300; 364/200;  
364/236.2; 364/248.1; 364/284; 364/284.2  
[58] Field of Search ..... 364/200, 300, 900  
[56] References Cited

U.S. PATENT DOCUMENTS

- 4,262,332 4/1981 Bass et al. .... 364/200  
4,649,473 3/1987 Hammer et al. .... 364/200

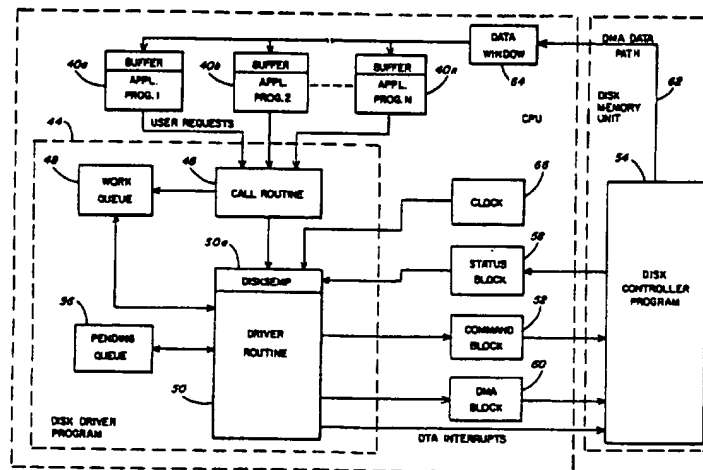
Primary Examiner—Raulfe B. Zache  
Attorney, Agent, or Firm—Wolf, Greenfield & Sacks

[57] ABSTRACT

A disk control system offloads to the disk controller

53 Claims, 24 Drawing Sheets

much of the overhead associated with disk operations and makes the CPU available for other work. A command block that fully specifies a user request for a disk operation is forwarded to the disk memory unit. The command block contains a unique identifier for tracking of user requests. User requests are executed by the disk memory unit in an order that is most efficient for the disk drive system. The status of a user request is communicated to the CPU via an interrupt and a status block containing the unique identifier. The status block indicates status conditions such as command read, completion and DMA channel request. The disk driver contains a work queue for user requests that have not been forwarded to the disk memory unit and a pending queue for user requests that are awaiting completion by the disk memory unit. By manipulation of the work queues and pending queues, the disk controller can be automatically reinitialized when an error occurs. The disk driver monitors the time that each user request is on the pending queue in order to detect failures of the disk memory unit.



## METHOD FOR DISK I/O TRANSFER

### FIELD OF THE INVENTION

This invention relates to a computer system having one or more magnetic disk storage units and, more particularly, to a method for efficient transfer of information between a central processing unit and the disk storage units.

### BACKGROUND OF THE INVENTION

Present day computer systems usually include one or more magnetic disk units for bulk storage. In a multiuser system, each user has the capability to read records from and write records to the disk units. Since disk operations take appreciable time in comparison with other computer operations, the efficiency with which disk operations are performed significantly affects overall system performance.

In general, the portion of a computer system associated with a disk unit includes the disk and associated drive hardware, disk controller software associated with the disk drive, disk driver software in the central processing unit and a communication channel between the disk driver and the disk controller. The disk driver receives requests from users via the operating system and communicates the requests through the communication channel to the disk controller. The disk controller receives the requests from the disk driver, provides the necessary signals to the disk drive hardware and returns information to the disk driver.

A user disk read or write request may involve several operations on a machine level. In prior art systems, the disk controller received only one instruction, such as SELECT, SEEK, READ or WRITE, at a time. Until the previous command was executed, the controller did not receive the next command. This mode of operation is inefficient because the CPU is required to perform several operations during which time other user disk requests are required to wait. In addition, the disk controller is not necessarily operating in the most efficient manner since it must execute each command in the order specified by the disk driver.

### SUMMARY OF THE INVENTION

According to the present invention, these and other objects and advantages are achieved in a method for executing user requests for work by a disk memory unit in a computer system including a central processing unit (CPU) having a main memory and a disk memory unit interconnected to the CPU. The method comprises the steps of; for each user request received, placing a queue block representative of the user request on a work queue; constructing a command block for each queue block placed on the work queue; forwarding each command block from the CPU to the disk memory unit for execution; after each command block has been forwarded to the disk memory unit for execution, transferring the associated queue block from the work queue to a pending queue; executing the user requests represented by each command block in an order determined by the disk memory unit; and removing each queue block from the pending queue after the disk memory unit has completed the associated user request. The command block forwarded to the disk memory unit contains all the information necessary for execution of a use request. The work queue and the pending queue maintain records of user requests received and user

requests forwarded to the disk memory unit for execution, while permitting control of disk memory operations to be transferred to the disk memory unit.

Each queue block in the work queue and in the pending queue and each command block has a unique identification number which links it to a user request. As a result, user requests can be executed by the disk memory unit in an order that is unknown to the CPU.

A status block indicative of the status of a user request is constructed by the disk memory unit and is forwarded to the CPU. The status block includes an interrupt type, the identification number received in the associated command, and status words which indicate errors and other information. The disk memory unit notifies the CPU via an interrupt and the status block when a command has been read. Each time the CPU is notified that a command block has been read, the work queue is checked for additional user requests. When additional user requests are found on the work queue, the CPU is notified to process the user requests and forward the required command blocks to the disk memory unit. As a result, user requests are quickly forwarded to the disk memory unit and the CPU is made available for other work.

The disk memory unit also requests the CPU via an interrupt and the status block to set up a data transfer channel when the disk memory unit is ready for transfer of data associated with a specified user request to or from the disk memory unit. After a user request has been completed by the disk memory unit, the CPU is notified via an interrupt and the status block that the user request has been completed.

According to another aspect of the invention, the time that each queue block has been on the pending queue is monitored and an error condition is generated when any of the queue blocks has been on the pending queue for longer than a maximum allowed time. The error condition causes the disk memory unit to be reinitialized. The work queue and the pending queue each have an associated head and a tail. Preferably, the step of reinitializing the disk memory unit includes the step of appending the queue blocks in the pending queue to the head of the associated work queue and initializing the pending queue to zero. As a result, user requests previously forwarded to the disk memory unit that would otherwise have been lost when the disk memory unit was reinitialized, are executed in the order that they were received. Reinitializing the disk memory unit includes the steps of converting from an intelligent mode of operation to a degraded, or dumb, mode of operation, downloading an intelligent mode disk controller program from the CPU to the disk memory unit and resuming operation in the intelligent mode. Downloading of the disk controller program and resuming operation in the intelligent mode can occur automatically without the knowledge of or intervention by the user.

The step of constructing a command block includes the step of writing command information to a specified location in main memory, and the step of forwarding each command block from the CPU to the disk memory unit includes the step of notifying the disk memory unit to read command information from the specified location in main memory. The step of constructing a status block includes the step of writing status information to a specified location in main memory, and the step of forwarding the status block from the disk memory unit

to the CPU includes the step of notifying the CPU to via an interrupt read the status block from the specified location in main memory.

The status block is forwarded to the CPU by means of a vectored interrupt. The interrupt sets a flag indicating that the status block has been forwarded to a specified location in memory and is available for access. The disk driver is able to identify spurious interrupts, to record an error and to continue operation. The disk memory unit has the capability of sensing specified internal and external events and error conditions unrelated to any user request and forwarding them to the CPU via the status block. The specified events are logged by the CPU, or processed in a predetermined manner. Unrecoverable errors result in the disk memory unit being automatically reinitialized as described above, and operation is resumed without intervention by the user.

The disk driver includes a trace function that can be selectively enabled for tracking system operations and errors. When the trace function is enabled, a real time entry is made in a trace table for each command block, each interrupt and each error that is handled by the disk driver. The trace table thus contains a traffic record useful for diagnostic purposes.

According to another aspect of the invention, there is provided a method for executing user requests for work by a disk memory unit in a computer system including a central processing unit (CPU) having a main memory and a disk memory unit interconnected to the CPU. The method comprises the steps of: for each user request, forwarding a command block that fully specifies the user request from the CPU to the disk memory unit, the command block containing a unique identifier; resuming normal operation of the CPU until notified by the disk memory unit of the status of the specified user request; executing the user request specified by the command block at a time determined by the disk memory unit; and notifying the CPU of the status of the specified user request via an interrupt and a status block containing the unique identifier. For user requests requiring data transfer to or from the disk memory unit, the method further includes the steps of notifying the CPU via an interrupt and the status block containing the unique identifier to set up a data transfer channel when the disk memory unit is ready for transfer of data associated with the specified user request, and transferring data associated with the specified user request to or from the disk memory unit under control of the disk memory unit.

### BRIEF DESCRIPTION OF THE DRAWINGS

For a better understanding of the present invention together with other and further objects, advantages and capabilities thereof, reference is made to the accompanying drawings which are incorporated herein by reference and in which:

FIG. 1 is a block diagram of a computer system incorporating the present invention;

FIG. 2 is a block diagram illustrating the interrelationship of software elements in the system of the present invention;

FIG. 3 illustrates the data structure of a command block in the system of the present invention;

FIG. 4 illustrates the data structure of a status block in the system of the present invention;

FIG. 5 illustrates the data structure of a DMA block in the system of the present invention;

FIG. 5A illustrates the format of an INITIALIZE AND INTERRUPT command;

FIG. 5B illustrates the format of a DOWNLINE LOAD command;

FIG. 6 is a flow diagram illustrating the general operation of the call routine in accordance with the present invention;

FIG. 7 is a flow diagram illustrating the general operation of the disk driver routine in accordance with the present invention;

FIG. 8 is a block diagram illustrating the structure of the work and pending queues in the disk driver;

FIG. 9 is a flow diagram illustrating the routine for processing a phantom interrupt;

FIG. 9A is a flow diagram illustrating the operation of the GET CONTROLLER STATUS routine;

FIG. 10 is a flow diagram illustrating the routine for processing a spurious interrupt;

FIGS. 11A and 11B are flow diagrams illustrating the routine for processing a COMMAND READ interrupt;

FIG. 12 is a flow diagram illustrating the routine for processing a TRANSFER REQUEST interrupt;

FIG. 13 is a flow diagram illustrating the routine for processing a BUFFER ACQUIRED interrupt;

FIGS. 14A and 14B are flow diagrams illustrating the routine for processing a COMPLETION interrupt;

FIGS. 15A and 15B are flow diagrams illustrating the routine for processing an ERROR interrupt;

FIG. 16 is a flow diagram illustrating the routine for processing an UNSOLICITED interrupt;

FIG. 17 is a flow diagram illustrating the routine for processing a RETURN CONTROLLER INFORMATION interrupt;

FIG. 18 is a flow diagram illustrating the routine for processing a user request;

FIG. 19 is a flow diagram illustrating the routine for processing a clock interrupt;

FIGS. 20A and 20B are flow diagrams illustrating the operation of the ABORT CONTROLLER routine; and

FIG. 21 is a simplified block diagram of a portion of the disk controller program.

### DETAILED DESCRIPTION OF THE INVENTION

A computer system incorporating the present invention is shown in block diagram form in FIG. 1. A computer 10 includes a central processing unit (CPU) and a main memory. A plurality of user terminals 20a, 20b-20n may communicate with and utilize the computer 10. The computer 10 is connected to one or more disk memory units. In the present example, the computer 10 is connected to four disk controllers 22, 24, 26, 28, each of which handles four disk drives 30, 32, 34, 36. The maximum number of disk drives per controller depends on the design of the disk controller. The system is designed to handle up to eight disk controllers, each having eight disk drives. In the computer 10, an operating system includes a disk driver program for each disk controller. Each disk controller 22, 24, 26, 28 includes a disk controller program which communicates with the respective disk driver program in the CPU 10.

The interrelationship of the disk driver program, the disk controller program for a single disk controller and user application programs is shown in FIG. 2. A plurality of application programs 40a, 40b-40n run in a time-shared manner on computer 10. When the application programs 40a, 40b-40n require work to be done by the disk memory units, user requests are forwarded to a disk

driver program 44 that is part of the operating system. The user requests are processed by a call routine 46 in the driver program 44. The call routine uses each request as a set of arguments to construct a user disk request queue block that is placed on a work queue 48. A driver routine 50 is notified by the call routine via a disk semaphore 50a that a user request has been received.

The driver routine 50 accesses user request queue blocks in the work queue 48 and constructs command blocks 52 that fully specify the user request. The command blocks 52 are forwarded to a disk controller program 54 in the disk memory unit. After the command block is forwarded to the disk controller program 54, the corresponding user request queue block is transferred by the driver routine 50 from the work queue 48 to a pending queue 56 until the user request is completed.

The disk controller program 54 communicates the status of the user request to the disk driver program 44 via a status block 58. The status block 58 is forwarded to the driver routine 50 by an interrupt which notifies the disk semaphore 50a. The status block 58 is used to indicate that a command block 52 has been read and, subsequently, the status block 58 is used to request setup of a DMA data transfer. The driver program 44 then forwards a DMA block 60 specifying the DMA channel information to the disk controller program 54. The actual data transfer occurs via a DMA data path 62 through a data window 64 in the CPU to a buffer in the specified application program 40a, 40b-40n. After completion of the data transfer, the disk controller program 54 notifies the driver program 44 via the status block 58 that the user request has been completed. A CPU clock 66 interrupts the driver routine 50 via the disk semaphore 50a at periodic intervals for timing each operation and generating an error condition when a timeout occurs. The necessary data structures and operating routines are described in detail hereinafter.

The purpose of the disk control subsystem of the present invention is to generally increase the overall throughput of disk I/O transfers. The efficiency of the disk subsystem is increased by the following features. (1) All pertinent information about each command is transmitted to the disk controller as soon as possible. This permits the disk controller to make intelligent decisions about how it will handle each and every command. (2) The queuing algorithm that governs the disk seeking order is performed on the disk controller. (3) Retry operations in reading or writing are performed by the disk controller rather than the disk driver program associated with the CPU. (4) Error correction is performed to the extent possible by the disk controller. (5) The disk controller can accept multiple commands for the same disk driver, thereby freeing the disk driver software for other operations in a shorter time. (6) The channel program overhead is offloaded to the disk controller.

The disk controller has the capability to receive and queue multiple requests. The controller performs its own SEEK optimization. Error correction is performed automatically after a prescribed number of unsuccessful attempts to read or write a record.

In order to initialize the disk control subsystem the disk controller program must be downline loaded to the disk controller. The operating system in the CPU reads the controller program from the system disks and sends it to the controller. When the entire program is sent to

the controller successfully, the downline load is complete.

The basic control sequence between the disk driver and the disk controller is through channel order packets of different types. The different channel order types are as follows: (1) initialize; (2) downline load; (3) command; (4) status; (5) DMA. Each of the channel orders is explained in detail hereinafter. The link between the driver and the controller is a series of command blocks sent to the controller by the driver and a series of status blocks returned by the controller to the CPU through the driver. The status blocks contain interrupt codes which are used to sequence disk requests to completion.

### COMMUNICATION PROTOCOL

The protocol for communication between the CPU and the disk controller is an asynchronous communication protocol. Commands and other parameters are passed by a non-interruptible, high speed data transfer (DMT) and data is transferred by direct memory access (DMA). The communication protocol is structured to allow the disk controller to obtain all the pertinent information about each command, such as READ or WRITE, at one time, and thereby allow the controller microprocessor to make intelligent decisions about how it will handle each command. The controller returns data to the CPU in record units only, except for status information.

The interface protocol consists of a handshaking sequence between the CPU and the disk controller under which communication and data blocks are passed between the two. Data words known as OTA's (output through register A) are used by the CPU to initiate an action by the controller, and interrupts are used by the controller to notify the CPU that an action has been completed. As noted above, the communication blocks used in the protocol fall into one of the following five categories: (1) initialize block; (2) downline load block; (3) command block; (4) status block; (5) DMA block. The initialize block and downline load block are used by the CPU in order to load the controller program into the disk controller and switch the controller from a non intelligent mode into the intelligent mode. The command blocks are sent from the CPU to the controller and define a command to be executed by the controller. The status blocks are sent from the controller to the CPU and contain status information. The DMA blocks are sent from the CPU to the controller and contain DMA channel and chain information to be used by the controller for DMA data transfer. All of these communication blocks are transferred between the CPU and the controller using DMT transfers.

### OTA's

The disk driver notifies the disk controller of required operations by using OTA's. There are five OTA's defined for this purpose, OTA 10, 11, 12, 13 and 00. The controller must be nonbusy to programmed input/output to accept an OTA. Since the controller can go busy at any time, the CPU must attempt an OTA and if the controller is busy, either constantly attempt the OTA or try again later.

An OTA 10, 11, 12 or 13 is used to send the starting address of a command block. On receipt of an OTA, the controller fetches the command block from memory. The controller remains busy while fetching the command block. After the command block is fetched, the controller sends a COMMAND READ interrupt ac-